Automatization in Lean

Damiano Testa

University of Warwick

RNTA mini symposium Atelier Lean 2023

May 2nd, 2023

Automation

Computers take on repetitive tasks.

In the context of formalization of mathematics, the computer also

- helps producing more complicated arguments, as it separates neatly different parts of the argument;
- informs, ideally, the **discovery** of new mathematical results;
- detects *very well* unnecessary hypotheses.

[The resulting generality is often only useful to simplify *formalization*, rather than *discovery* of mathematics.]

Currently, Machine Learning, Artificial Intelligence, Neural Networks and auto-formalizations are not yet really available.

There is lots of interest and steady progress on this front.

Tactics

Any tactic is a form of automation.

Tactics allow to maintain abstraction:

- we humans talk about mathematical concepts,
- the computer has some representation for these concepts.

Tactics bridge this gap.

We do not need to know what the computer's internal representation is: tactics handle the translation.

In the previous talks, you have already seen some tactics (`exact`, `intro`, `apply`, `rw`, ...).

Now, we talk about `library_search` and `simp`.

These tactics probably feel closer to an intuitive idea of “automation” that you may have.

`library_search`

`mathlib` is a massive repository: it contains

- over 1 million lines of code
- over 60 thousand lemmas.

Most of the basic[1] lemmas are already available.

`library_search` helps you find them!

[1] "Basic" may mean *really* basic, to a level that you may not even consider them "lemmas".

```
import tactic

example {a b c : ℕ} : a ^ (b + c) = a ^ b * a ^ c :=
by library_search

--  Try this: exact pow_add a b c
```

Click here to open the Lean web editor.

Besides `library_search`, `mathlib` has a very helpful naming convention that allows you to “guess” names of lemmas.

The simp-lifier

As the name suggests, the simp-lifier tries to simplify a goal.

```
import tactic

example {a b : ℤ} :
  - (-1 * a + 0 * b) = a * (1 + a * 0) :=
begin
  simp,
end
```

Click here to open the Lean web editor.

simp automatically used the lemmas

neg_mul	neg_neg	add_zero
one_mul	mul_one	
mul_zero	zero_mul	

“simp-lemmas”: lemmas that simp uses

- They assert an equality or an iff.
- The LHS **looks more complicated** than the RHS.

```
#print one_mul    -- means:   1 * a = a
#print zero_mul   -- means:   0 * a = 0
#print add_zero   -- means:   a + 0 = 0
#print neg_neg    -- means:    - -a = a
#print mul_zero   -- means:   a * 0 = 0
#print mul_one    -- means:   a * 1 = a
#print neg_mul    -- means:  -a * b = -(a * b)
```

The asymmetry helps Lean: it flows along

$$\texttt{hard LHS} \longrightarrow \texttt{easy RHS}.$$

[Being a “simp-lemma” is something that *you* must communicate to Lean: there is no automated mechanism that makes Lean self-select which lemmas are simp-lemmas.]

Let's switch over to an interactive demo.